Papers from the AAAI Fall Symposium (FS-09-02)

Managing Conversation Uncertainty in TutorJ

Vincenzo Cannella and Roberto Pirrone
Dept. of Computer Science and Engineering (DINFO)
University of Palermo
Viale delle Scienze Ed. 6
90128 Palermo, Italy
{cannella,pirrone}@dinfo.unipa.it

Abstract

Uncertainty in natural language dialogue is often treated through stochastic models. Some of the authors already presented TutorJ that is an Intelligent Tutoring System, whose interaction with the user is very intensive, and makes use of both dialogic and graphical modality. When managing the interaction, the system needs to cope with uncertainty due to the understanding of the user's needs and wishes. In this paper we present the extended version of TutorJ, focusing on the new features added to its chatbot module. These features allow to merge deterministic and probabilistic reasoning in dialogue management, and in writing the rules of the system's procedural memory.

Conversational agents are communication technologies enabling an interaction based on natural language. Chatterbots, or chatbots are typical examples of such systems. Eliza (Weizenbaum 1966) or Alice (ALICE 2009) are probably some of the most famous ones. In spite of the fact that they were developed with different technologies, they both have been designed as simple stimulus-response systems, without a state management, or interaction planning. Moreover they are designed only for a linguistic interaction. Natural language is probably one of the most intuitive interaction modality. Many efforts have been made to make computers able to use natural language but they are very far from being able to speak fluently, and to face every possible subject during the dialogue with complete understanding of the user's sentences. Finally, stimulus-response chatbots are not able to modify their behavior according to specific characteristics of each user. To reach this goal, they should be able to build an internal representation of the user.

We already presented the cognitive architecture of TutorJ, an Inteligent Tutoring System inspired to the Human Information Processor Model (HIPM) that is one of the main models developed to describe human cognitive processes. According to HIPM, a cognitive architecture is composed by a series of modules. A module carries out a specific function involved in the cognitive processes. Modules are arranged as perceptual, cognitive, and motor ones. A complete description of our cognitive architecture can be found in (Pirrone, Cannella, and Russo 2008). In this work we focus on a specific TutorJ component, the Chatbot module that manages the whole dialogue. The Chatbot module uses a part of the long-term memory module, which manages the procedural knowledge of the system. Part of this knowledge describes natural language interaction structures as suitable rules written in Artificial Intelligence Markup Language (AIML) (AIML 2009). Moreover, the Chatbot module's working memory stores the stimuli from the environment, together with the status of the conversation. The chatbot interface interacts with the user using natural language, as the sensor/motor module of the system. We can say that our chatbot includes all the cognitive parts of TutorJ and has the responsibility to coordinate the rest of the system. The Chatbot module has been derived from the well-known chatbot A.L.I.C.E. In our work we have extended it, designing a more general architecture for conversational agents. Our system is able to merge linguistic and graphical interactions, and allows the programmer to choose the most suitable interaction modality. We do not use a simple stimulus-response architecture. The chatbot can be programmed to reach a specific goal. The programmer can design different possible behaviors for different goals. Alternatively, the system can be provided with a collection of many possible actions, and it can choose among them autonomously. Finally, the engine of the new version has been extended including probabilistic reasoning, to cope with uncertainty in the dialogue, and a suitable connection has been added with external knowledge repositories like OWL ontology and WordNet. In this way standard symbolic reasoning is allowed. Fig. 1 reports a simple outline of the the new Chatbot module, and its connection with the new components. We present the new Chatbot module as a general framework where deterministic and stochastic behavior can be merged. A programmer can define more sophisticated chatbot agents in a very simple manner with a a lower effort, obtaining a richer interaction. The rest of the work is arranged as follows. Next session will be devoted to describe the state of the art about conversational agents. Then the extensions to the conventional chatbot architecture, and the new AIML tags for integration of Bayesian networks will be presented. Fourth section will describe the use of Bayesian reasoning through *Partially Observable Markov Decision Processes* (POMDP). Finally, the connection with externall knowledge repositories will be presented, and some conclu-

sions will be drawn.

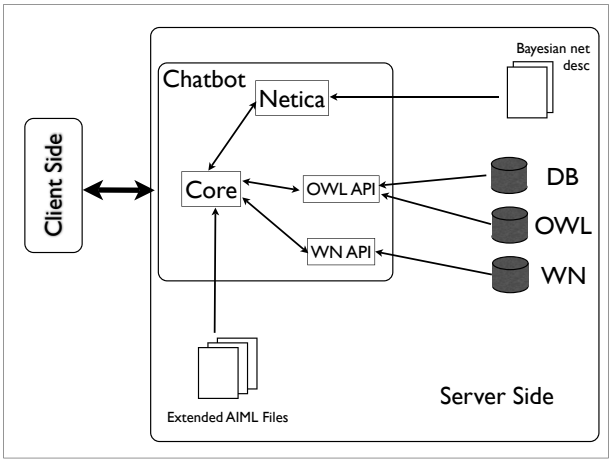

Figure 1: An overview of the Chatbot module. The Core component implements the extended chatbot engine using the new AIML tags to cope with Netica, OWL, and WorNet API.

State of the Art

AIML is one of the most popular languages to program chatbots' knowledge bases. AIML stands for Artificial Inteligence Markup Language. It is an XML based language. The system is defined as a stimulus-response agent. The interaction is defined by a set of two-way moves, composed by a sentence of the user, and the corresponding remark of the system. Each adjacent pair in the dialogue is defined in a `<category>` element, which includes `<pattern>` and a `<template>`. Patterns represent stimuli from the user, while templates are the corresponding system replies.

To control the flow of the whole dialogue, the system should manage the context of the conversation. To this aim, AIML has the `<that>` tag to link a category to the previous interactions between the user and the system. The `<template>` element can be constructed merging more categories together, using the `<srai>` tag that allows referring to other categories. Categories can be organized according to topics, through the `<topic>` tag. This tag can contain many category tags together. All of them treat the same topic. The following example shows a simple couple of sentences, related to the context by the `<that>` tag.

```
<category>
<pattern>Yes, I do</pattern>
<template>
<that>Do you like music?</that>
Which artist, in particular?
</template>
</category>
```

AIML is used to program A.L.I.C.E., which was inspired to the famous test proposed in 1950 by A. Touring. The interaction with A.L.I.C.E. is only text-based. It uses a simple interface similar to that of a chat. Other chatbots are included into more advanced 3D graphical systems, as avatars in virtual worlds, or they are able to perform speech analysis and syntesis. In the past, we presented Graphbot (Pirrone et al. 2008) that is an enhancement of the A.L.I.C.E. base system, and allows a more complex interaction through WIMP 2D user interfaces. We called GAIML the Graphbot extended AIML language. This system is able to modify its own graphical interface according to the specific interaction with the user. Interfaces proposed during the interaction can be defined directly by the programmer in the `<template>` element related to the `<category>` involved in the interaction step. On the contrary, the system can generate autonomously the interface analyzing the content exchanged with the user. In the second case, a set of rules define correspondences between data patterns and interface patterns. Interfaces are generated on the basis of these correspondences.

Another interesting example of extention of AIML is iAIML, presented in (Neves, Barros, and Hodges 2006). iAIML has been developed to include intentions in categories. The authors have been inspired by the linguistic Conversational Analysis Theory, which treats intentionalities in adjacent pairs in dialogue. The intentional information is expressed through three variables: session, user intention and bot intention. Values assigned to these variable characterize the organization of the conversation, subdivided into three main parts: opening, development, and closing. In this way, the system can track the phase gained by the conversation. Categories can be classified on the basis of the part of the conversation in which they should be used. On the basis of this new classification, the categories of A.L.I.C.E. have been re-classified. This approach resulted more consistent then those applied in the past.

The approach adopted in AIML systems is prone to errors due to the uncertainty in understanding the user input. A pattern-matching approach is too limited without a deep analysis of the input. On the other hand, besides to the successes in Natural Language Processing, we are very far from a complete and affordable understanding of whatever utterance. Uncertainty is often managed using stochastic systems, and such techniques have been largely applied to manage dialogues in natural language too. Bayesian networks are one of the most used technologies in this field. The reader is referred to (Horvitz et al. 1998) and (Kim, Hong, and Cho 2005) as an example. In both works, authors describe the use of Bayesian networks to model either the internal state of the user or the dialogue. They use these networks to analyze the dialogue, the goals of the user, and the right meaning of the users sentences. Some other works used stochastic techniques, not only to analyze data from the interaction, but to plan the same interaction too. To this aim, researchers have investigated both Markov Decision Processes (MDPs) and Partially Observable Decision Markov Processes (POMDPs).

In (Diane et al. 2002) there is a first example of MDP applied to dialogue management. The limitations of MDPs are related to the fact that they consider the state of the dialogue totally available. This is hardly true in natural language interaction. Some researchers tried to solve the problem with POMPDs.

A first attempt was in (Diane et al. 2002). The proposed

POMPD managed discretized variables but do not define how to optimize the discretization. (Zhu and Zhao 2002) proposed the use of Bayesian networks to combine information from many continuous and discrete sources to compute probabilities. In (Roy, Pineau, and Thrun 2000) a comparison between a MDP version and a POMDP one of the same dialogue system is presented. The result is that the POMDP version gains more reward per unit time than the MDP version. Finally (Williams and Young 2005) presents a dialogue system developed with a a factored POMDP. In this model, the POMDP state variable is separated into three unobservable conditionally independent components: the users goal, the users action, and the history of the dialogue. The independence hypothesis allows an high computational cost reduction.

Extensions in the Chatbot module

The first version of TutorJ was a stimulus-response system. The chatbot gave replies to an input from the user in a predefined manner. In this way, the system was not able to plan the interaction sequence. A simple mechanism had been implemented to define a context as the sequence of most recent sentences exchanged by the user and the system. There was no explicitly defined goal even if past conversation could be taken into account while choosing the next dialogue move. Obviously, some AIML code could have been written to implement categories inspired to some possible goals. In this way, goals were not managed directly by the system itself. We extended the Chatbot module to overcome the limitations outlined previously, and to obtain a new kind of procedural memory for the chatbot where dynamic AIML rules encoding strategies to attain a goal could be stored. We distinguish two kinds of goals: the *immediate goal*, and the *final goal*. The former has to be reached in a single step; the latter can be pursued within the whole conversation. We had to change both the core chatbot system, and the AIML language. The extended AIML includes also all the GAIML tags. The system maintains an internal string variable to store the immediate goal. A similar variable stores the type value. To this aim, the system has to establish if a certain action can be used to reach the goal. Actions are described in an AIML file, and they could be tagged on purpose with such an information. As a consequence the AIML structure needs to be changed. The language has been extended, and the new tag `<goal>` has been defined. This element is a child of the `<category>` tag along with `<pattern>` and `<template>`. An agent will perform the action described in the `<template>` to reach the particular goal contained in the new element. This is a simple way to introduce Pragmatics in the AIML description of dialogues. Each sentence is regarded as an act with its specific goals in the conversation. Pragmatics has proposed several possible goals for interaction acts. Such goals have been organized into classes and hierarchies too. We have defined a generic framework that allows the designer to adopt any possible collection of goals. Many different categories can share a common pattern associated to different goals. In this way the behavior of the system can be adapted to the different goals using different templates. A category can have no goal element. In this way, old AIML code is still supported. The system's goal can be changed during the interaction. The change can be made by code. Some new tags have been defined to manage directly the goal of the system. These tags can be included into the template element of a category. They allow to read or change the value of the goal. The introduction of the `<goal>` element has inspired another sibling: `<type>`. This tag allows to assign a generally defined typology to the category. This tag has been defined to cover many different needs. It can be used, for instance, to change the behavior of the chatbot. Interactions can be grouped on the basis of the mood or the nature of the chatbot related to their templates. The tag `<type>` can be used to distinguish interaction modes, as linguistic or graphical ones. In this way, the AIML programmer can define different possible templates for a same pattern, distinguishing between them.

Bayesian GAIML

The introduction of tags `<type>` and `<goal>` allows the programmer to group categories on the basis of the way she wants to use their templates. However, the system is not able to plan its behavior yet. This work is made by the programmer, who assigns each category to a particular goal, and establishes system's reaction to an input when a specific goal has to be reached. Moreover, this is only an immediate goal. There is no management of the whole conversation with respect to a final goal. This remains, essentially, an advanced version of a stimulus-response agent. Another problem is related to the deterministic behavior of the system. In particular, both the `<pattern>` and the `<template>` element have to be remarked in this respect. A pattern defines the possible input from the user but the natural language interaction is prone to many possible mistakes. Assessment of the user input can lead to wrong or incomplete understanding. In a past work some of the authors tried to enlarge the abilities of the system in understanding the user sentences through the LSA technique (Pilato, Pirrone, and Rizzo 2008). This solution has improved the performance of the system, but didn't solve the problem. A more detailed matching has been obtained with this approach. The input is no more treated as a textual pattern but it's deeply analyzed with respect to the content, trying to gather the meaning of the sentence. The results of this analysis can be wrong or incomplete too. For this reason, we moved towards probabilistic matching where not only the result is returned but also its probability. Similar considerations are possible for the actions of the system written in a `<template>`. Ideally, the system takes an action as a reply to an input, and to reach a specific goal but the flow of the conversation cannot be decided a priori. The system chooses an action because it will have certain effects, but these effects are not guaranteed. For instance, if the system says something to the user, she couldn't understand it. Conversation planning can not ignore both these problems. The system must to consider the inaccuracy in perceiving the world and the uncertainty of the results of its actions. For these reasons we have extended the system, defining a new set of AIML tags to interact with a Bayesian network, implemented with Netica (Netica 2009).

AIML Tags for the Bayesian Module

Netica is a complete development environment to build belief networks and influence diagrams. Several new AIML tags have been defined to allow the integration between the Netica module and the rest of the system. All of them allow to program the chatbot as Bayesian system. They allow tight interaction with an external Bayesian network loaded and managed by the Netica module.

In general, the chatbot interacts with the Bayesian network passing it the stimuli from the environment, and obtaining useful information to establish the next action. The network can be used to set the values of the `<type>` and `<goal>` slots, thus guiding the flow of the dialogue. The system can manage a list of nets. The `<loadBN>` tag loads a single Bayesian net, assigning it a name. This name is then used by the code to interact with the net. When a fact has to be asserted, one has to specify which net has to receive this information to process it. Moreover, the fact refers to a specific node in the net that must be specified too. To this aim, each node of the net has to have an identifier. The tag used to assert a fact is the `<enterState>` tag. On the contrary, a fact can be denied. To this aim, we have defined the `<enterStateNot>` tag. The net can be reset using the `<retractFindings>` tag. In this way, it becomes as just loaded. Finally, the network can be deleted using the `<delBN>` tag. Standard AIML tags can be nested into the Netica ones to guide the computation when the system reaches a new state. After having asserted or denied something, the chatbot can use the network to evaluate the probability of some particular event. To this aim, it can use the `<getBilieaf>` tag. Besides managing belief networks, Netica allows to manage decision networks too. In this case, the chatbot can use the network to establish which is the best next action. two possible tags are at disposal in this case. The first one is the `<getExpectedUtils>` tag, which returns the estimated rewards for all the possible actions. The `<getBestDecision>` tag is used if the programmer wants the network to return only one action that is the best one. The choice is made by the network. The tags just described are the most important ones implemented to manage the interaction with Netica. The whole set of tags offers a generic framework to merge the functionalities of a chatbot and of a Bayesian network. Any possible Bayesian network can be loaded, and managed by code in the AIML categories.

Partially Observable Markov Decision Process Module

The use of Bayesian networks can help to manage the uncertainty in a conversation. The problem of planning the whole conversation to reach a final goal can be solved using a Partially Observable Markov Decision Process Module (POMDP). A POMDP is described as a tuple (S, A, T, R, O, Z), where:

- S represents a finite set of states
- A represents a finite set of actions
- $T : S \times A \times S \rightarrow \Pi(S)$ is the state transition function where $T(s, a, s')$ is the probability of passing from state s to te state s' when the action a has been executed in the state s
- $R : S \times A \rightarrow R$ is the reward function and $R(s, a)$ is the expected reward for taking an action a when the system is in the state s
- O represents a set of observations
- $Z : O \times S \times A \rightarrow \Pi(O)$ is the observing function and $Z(o, s, a)$ is the probability function of observing o when the system is in the state s after having made the action a

At each step, the machine is in some state s that isn't known exactly. In fact, s can not be directly observed. For this reason, a distribution b over the states is maintained that is called belief state. The value $b(s)$ represents the probability of being in a specific state s. The system takes the actions maximizing the expected reward over a number of decision epochs or events that can be also unlimited. The number of considered epochs represents how far is the so-called horizon of the system. In our work, we consider a finite horizon. After an action, the system observes the world, registering the value o. As just stated previously, this value depends on the last action a, and the actual state s. At this point, the system must evaluate the actual state. This evaluation is given by

$$b'(s') = \frac{O(o', s', a) \cdot \sum_{s \in S} T(s', a, s)}{Pr(o|a, b)}$$

where denominator is a normalizing term.

When the number of epochs is finite, then the agent acts to maximize the expected reward over a finite horizon. If k is the length of the horizon, and r_t is the reward gained at the t-th step, the reward is computed as

$$E[\sum_{t=0}^{k-1} r_t]$$

The system has to compute the optimal sequence of actions, maximizing this value. A t-step sequence can be usually represented through a tree structure. Let p be a 1-step decision tree, corresponding to a one action sequence. The gain for this sole action is equal to

$$V_p(s) = R(S, a(p))$$

where $a(p)$ is the action executed at the root of the tree. Generally, for a t-step tree, the gain is computed recursively. Let be $o_i(p)$ a $(t-1)$-step tree associated with the observation O_i. In this case, the gain is equals to

$$V_p(s) = R(S, a(p)) + \\ +\gamma \sum_{s' \in S} T(s, a(p), s') \sum_{o_i \in O} O(s', a(p), o_i) V_{o_i(p)}(s')$$

Netica is not able to manage a POMDP: it implements only simple Bayesian networks. We extended the interaction module between the chatbot and Netica to allow man-

aging complex POMDPs. A simple network can be used to implement a single step POMDP. The module manages the sequence of steps, using the value computed by the Bayesian network at a certain step to supply the input to the same network for the subsequent step. Moreover, the module computes the best sequence of actions to reach the goal. The module manages only a finite horizon. The number of steps is a tunable parameter. The flow control is shared among the deterministic control of the AIML code, and the probabilistic one of the Bayesian module. Sometimes the latter one does not choose actions, but only assesses the state, and sets the immediate goal or the type of the interaction, thus giving only a generic direction towards the goal. This information is then applied to the most suitable `<category>`.

Connection with knowledge repositories

The flow of the dialogue can be guided by deterministic reasoning too. To this aim, the module has been extended to manage the declarative knowledge of TutorJ about its domain that is stored in ontologies. The general data structure of a common ontology is a graph whose nodes represent concepts, and arcs represent relations among them. We provided the chatbot with a suitable set of AIML tags for connection to a generic OWL ontology and to WordNet.

The interaction with an OWL ontology

As previously explained, a chatbot is usually a stimulus-response system. Its behavior is dictated by what the user writes. In this way, a chatbot can be hardly programmed on the basis of a wider goals, but a direct reply to the user. As side effect, the conversation is not explicitly guided on the basis of topics. To solve this problem, we have enlarged the flow control model of the chatbot, that is guided by the relations between topics of the treated domain. Either concepts and relationships between them can be described with an ontology. OWL (W3C 2004) is an XML-based language widely used to describe an ontology. In the past, the inclusion of an ontology had been just investigated in TutorJ using the Cyc framework (Reed and Lenat 2002). The Chatbot engine in TutorJ has been modified to interact with OWL ones. The tags devoted to Cyc management have been ported to OWL technology. Cyc is a more complex technology, including an inference engine too. In the case of OWL, the role of inference engine is usually played by Jess. Till now, TutorJ does not use Cyc's inference engine. It browses autonomously the ontology, and moves between concepts following the relations among them. Ontology browsing is coded in AIML. We have developed suitable tags to reach this goal. These tags allow to load an OWL ontology, to retrieve the properties of a concept, and to move along relations. Developed tags reflect common OWL APIs. The main interactions to get data from an ontology have been implemented. On the contrary, functionalities useful to modify the ontology have been neglected. The knowledge of the system is considered static. The ontology can be modified only by the botmaster. Each node in an OWL ontology can be linked to external contents too. Each node is linked to a fragment of text too. the name of the node is meaningless, and this fragment can be used to assemble sentences parametrized with the content of this fragment. It can link also to a node of the Bayesian net managed by the system. When the system assesses a certain event as a likely one, or it selects an action as the best one, the link between the corresponding node in the Bayesian net, and a node in the OWL ontology would allow other possible inferences. The relationship between the ontology and the Bayesian net can be used on the other side. At first, the system browses the OWL ontology, then it follows the link from a node of the ontology to a node of the Bayesian network to evaluate a stochastic variable related to it. As an example, let's consider a bayesian network whose nodes represent the concepts in the ontology. Relations between nodes describe the conditional probabilities that the user can be interested in a concept, if she is in a related one. In this way, the same control flow of the conversation is guided by the ontology in a probabilistic way.

The interaction with WordNet

Uncertainties and difficulties in conversation are often due to the domain-specific terms. In these cases, the programmer can take advantage of the interaction between the chatbot and WordNet. We have implemented a collection of tags to interact with WordNet. These tags can be used to issue any possible query to the thesaurus. The system can browse the net walking the relations between the words, and its possible to retrieve words linked by synonymy, hypernymy, and hyponymy relations. WordNet is used to try disambiguation of the terms put in a sentence by the user to help context analysis when standard assessment fails. Sometimes the system has analyzed the user input, and it isn't able to get the meaning of a term surely. Then it turns to WordNet to get all possible meanings of that term, while asking the user to specify the exact word she intended. The system lists all possible meanings with a combo box list, and the user checks the suitable combo. WordNet is used also to get hyponyms and hypernyms of a term. In this case, the system can be used by the chatbot as a way to guide the conversation. The choice of a term in a sequence of terms related by hyponymy/hypernymy relations, is dictated by the degree of specificity of the used lexicon. A too specific term could not be understood by the user. If the user asks the meaning of a term, and the explicit definition of the term is not enough, the system tries to use a more simple term. In this case, it will use the hypernym of the not understood term. Synonymous words can enlarge the range of search, including new topics not considered by the user, and stimulating the conversation. WordNet allows the system to retrieve antonyms of the word, and inflected variants as plurals or different conjugations. The new implemented tags reflect the functionalities of the WordNet API, enriched with some formatting functions to adapt the content retrieved from WordNet to the conversation. The tags, after having retrieved the content, return the reply sentence, or a piece of text to be inserted in a wider sentence. The content can be presented in a textual form, or it can be used to compose dynamically graphical interfaces. For instance, when the system wants to list the possible meaning of a word, it can use a form with a list of radio buttons where the user can select the intended mean-

ing. Obviously, this solution is possible when the <type> element of the related <category> allows the graphical interaction. In this way, the value of the <type> tag controls the behavior of a tag, not only the flow of the conversation.

Conclusions and Future Works

A new framework for a conversational agent enabled with uncertainty management has been presented in this work as a new module of TutorJ that is a cognitive architecture for tutoring purposes. The new generation of educational systems will be equipped with meta-cognitive abilities to support the student in her learning tasks. We claim that the linguistic ability is the core technology for the evolution of present ITSs so they must exhibit fine interaction in natural language. As a consequence, all the ambiguities during the dialogue with the student due to misunderstanding or incomplete sentences, must be overcome. This is the motivation of our work in a nutshell.

To achieve our objective, the new version of the TutorJ chatbot allows to merge together deterministic and stochastic control of the flow of the dialogue. We have largely extended the AIML language, adding new tags to program the knowledge base of the Chatbot module, and enriching the procedural memory of the system with Bayesian reasoning.

The new chatbot architecture is general, and can be used apart from TutorJ. Future developments of this work will be in the direction of integrating our conversational agent in a more general cognitive model aimed to describe a conversation with a student, which is partially aware of what she wants to know even if she's posing precise questions about a topic.

References

AIML. 2009. Arttificial intelligence markup language. http://www.alicebot.org/documentation.

ALICE. 2009. Alice. http://www.alicebot.org/.

Diane, S. S.; Litman, D.; Kearns, M.; and Walker, M. 2002. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *Journal of Artificial Intelligence Research* 16:105–133.

Horvitz, E.; Breese, J.; Heckerman, D.; Hovel, D.; and Rommelse, K. 1998. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Fourteenth Conference on Uncertainty in Artificial Intelligence*, 256–265. Morgan Kaufmann.

Kim, K.-M.; Hong, J.-H.; and Cho, S.-B. 2005. Intelligent web interface using flexible conversational agent with semantic bayesian networks. In *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*, 313. Washington, DC, USA: IEEE Computer Society.

Netica. 2009. Netica bayesian network software from norsys. http://www.norsys.com/.

Neves, A.; Barros, F.; and Hodges, C. 2006. iaiml: a mechanism to treat intentionality in aiml chatterbots. In *Tools with Artificial Intelligence, 2006. ICTAI '06. 18th IEEE International Conference on*, 225–231.

Pilato, G.; Pirrone, R.; and Rizzo, R. 2008. A kst-based system for student tutoring. *Appl. Artif. Intell.* 22(4):283–308.

Pirrone, R.; Russo, G.; Cannella, V.; and Peri, D. 2008. Gaiml: A new language for verbal and graphical interaction in chatbots. *Mob. Inf. Syst.* 4(3):195–209.

Pirrone, R.; Cannella, V.; and Russo, G. 2008. Awareness mechanisms for an intelligent tutoring system. In *Proc. of 23-th AAAI Symposium on Biologically Inspired Cognitive Architectures*.

Reed, S. L., and Lenat, D. B. 2002. Mapping Ontologies into Cyc.

Roy, N.; Pineau, J.; and Thrun, S. 2000. Spoken dialogue management using probabilistic reasoning. In *ACL '00: Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, 93–100. Morristown, NJ, USA: Association for Computational Linguistics.

W3C. 2004. Owl web ontology language use cases and requirements. http://www.w3.org/TR/webont-req.

Weizenbaum, J. 1966. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM* 9(1):36–45.

Williams, J., and Young, S. 2005. Scaling up pomdps for dialog management: The "summary pomdp" method. In *Automatic Speech Recognition and Understanding, 2005 IEEE Workshop on*, 177–182.

Zhu, X.-Z., and Zhao, J.-M. 2002. Spoken dialogue management as planning and acting under uncertainty. In *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*, volume 2, 669–673 vol.2.